AVR: Abstractly Verifying Reachability

Aman Goel and Karem Sakallah

University of Michigan, Ann Arbor
{amangoel,karem}@umich.edu

Abstract. We present AVR, a push-button model checker for verifying state transition systems directly at the source-code level. AVR uses information embedded in the word-level syntax of the design representation to automatically perform scalable model checking by combining a novel syntax-guided abstraction-refinement technique with a word-level implementation of the IC3 algorithm. AVR provides independently-verifiable certificates that offer provable assurance and are easy to relate to the word-level system. Moreover, proof certificates can be further used in innovative ways to extract key design information and are useful in a growing number of applications.

1 Introduction

Model checking [24, 25] techniques based on incremental induction (like IC3 [16, 29]) have gained significant success [18] due to their property-directed nature and clever use of incremental SAT solving. Bit-level implementations of IC3, however, struggle with scalability due to being overwhelmed by low-level propositional learning [31]. Rapid advances in SMT solving [10] offers a solution and allows for performing IC3 directly at the word level by combining the incremental induction algorithm with an abstraction-refinement procedure [15, 38, 20, 32].

AVR is a model checker for verifying *safety* properties directly at the *first-order logic* using SMT solving, primarily designed for hardware verification. AVR uses *syntax-guided abstraction* [32], a generalization of implicit predicate abstraction [19], to perform IC3-style reachability analysis using word-level clause learning. The verification results generated with AVR are accompanied by independently-verifiable certificates i.e. inductive proof when the property holds, and counterexample execution trace when the property violates. These certificates can be externally verified using a proof checker or a trace simulator to offer provable security and can further be used in innovative ways (like for verifying distributed protocols defined over unbounded domains [41, 42]). AVR also provides a variety of complementary verification techniques for scalability (like data abstraction, interpolation), as well as useful utilities (like design statistics, graphical visualizations) to allow high-level insights on the input design.

2 Motivation

Consider a predicate $p := (a + b < 1)$ defined over two 32-bit variables a and b. An equivalent propositional-level representation of p will involve a bit-blasted

expression involving 64 Boolean variables and several hundred clauses. As a consequence, bit-level model checking algorithms struggle with *state-space explosion* [23] with increasing bit-width of variables.

AVR derives its motivation from the fact that the word-level representation of a problem contains useful high-level information that can be exploited for better scalability. AVR uses this insight to infer an implicit syntax-guided abstraction using terms built from objects present in the word-level syntactic description of the problem (like a, b, 1, $+$, $<$). The approach can be further combined with data abstraction using uninterpreted functions [17, 9] to simplify reasoning for the underlying query solver. This, coupled with efficient SMT solving, allows for an effective word-level model checking algorithm that can scale better than bit-level engines for a variety of verification problems [31, 32]. Moreover, the induction-based procedure has the unique strength of producing word-level proof certificates (aka *inductive invariants*) that are useful in a variety of applications [30, 34, 42, 41].

3 System Architecture

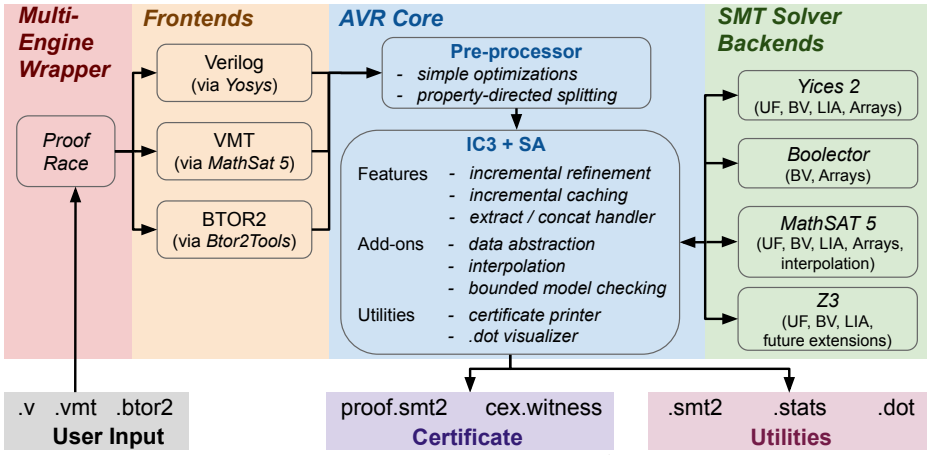

Fig. 1: Verification flow with AVR
UF: uninterpreted functions, BV: bit-vectors, LIA: linear integer arithmetic

Fig. 1 shows the architecture and verification flow with AVR.

Frontends in AVR extract the model checking problem from inputs in different formats using openly-available tools.

- Verilog + SystemVerilog assertions [7] (using Yosys [50])
- VMT [6] (using MathSAT 5 [21])
- BTOR2 [47] (using Btor2Tools [2])

AVR core performs IC3 with syntax-guided abstraction (IC3+SA) and implements several verification techniques and utilities (detailed in §3.1, §3.2).

SMT solver backends use the latest versions of state-of-the-art SMT solvers (Yices 2 [28], Boolector [46], MathSAT 5 [21] and Z3 [26]) to efficiently integrate incremental solver reasoning with *AVR core* using a C++ interface.

Multi-engine wrapper allows for process-level parallelism by running multiple instances of AVR in parallel using *proof race* (as elaborated later in §3.3).

3.1 Techniques

At its core, AVR implements a word-level IC3 procedure where terms in the implicit syntax of the problem are used as building blocks to perform IC3-style clause learning at the word level using SMT solving. The key differences between IC3+SA [32], as implemented in AVR, and bit-level IC3 [16, 29] can be summarized as follows:

- IC3+SA uses relations defined over syntax terms (referred as *atoms*) instead of individual state bits to implicitly represent an abstract state space.
- SMT solving is used instead of propositional SAT solving for solver reasoning.
- Counterexample-guided abstraction refinement [22] is used to automatically eliminate the spurious behavior in the syntactically abstracted domain by identifying new terms from the proof of unsatisfiability [39].

Within the core IC3+SA framework, AVR implements several optimizations and important features that are helpful in improving model checking performance.

Core features

- *Pre-processor optimizations* perform simple transformations to standardize and optimize the input model extracted from different input formats.
- *Incremental refinement* performs abstract counterexample analysis in an incremental fashion by using single-step solver queries instead of conventional multi-step path queries.
- *Incremental caching* allows caching frequently-used data structures to speed up incremental SMT solving (at the cost of increasing memory usage).
- *Multiple SMT backends* allow configuring usage of different SMT solvers for different kinds of SMT queries based on the type of query.

Add-on techniques

- *Property-directed splitting* breaks wide words at bit-field extraction and concatenation boundaries [8] in a property-directed manner.
- *Data abstraction* combines IC3+SA with data abstraction using uninterpreted functions [17, 9, 38] by converting data operations as uninterpreted, and allows focusing on the control structure of the problem.
- *Interpolation* adds using Craig interpolants [43] with incremental refinement to extract new terms from a spurious abstract counterexample.
- *Extract / concat handler* adds a novel dedicated engine to deal with lightweight interpretation of bit-field extraction and concatenation operations.
- *Bounded model checking* (BMC) [12] allows for an alternative to the IC3+SA engine for quick bug hunting, especially for detecting shallow bugs.
- *Other options* include adding global assumptions *lazily*, minimizing proof certificates, making syntax-guided abstraction closer (resp. farther) to implicit predicate abstraction by decreasing (resp. increasing) abstraction *fineness*, exploiting *randomness* during solving, and a few others.

Utilities

AVR also provides a number of useful utilities to the user including:
- Printing the problem in SMT-LIB format [10].
- Graphical visualizations of the problem and the word-level clause learning.
- Detailed statistics report on the input design and the verification run.

3.2 Certificates

Once a model checking problem is solved, there can be two possible outcomes: either the property holds, or the property can be violated.

If the property holds, IC3+SA produces an inductive invariant, i.e. an approximate fixpoint that establishes the property to be true in *all* executions of the system. Inductive invariants act as proof certificates that guarantee the correctness of the verification outcome. AVR prints such proof certificates directly in the SMT-LIB format [10], which allows for independent checking of their correctness using an external SMT solver. Since proof certificates are in the word-level format, they are human-readable and much easier to relate to the word-level input directly at the source-code level (as against bit-level invariants which are usually too hard to understand). Proof certificates find their applications in different ways, like to derive inductive validity cores [30] to gain design insights, to derive assume-guarantee verification conditions [34, 49], to automatically derive helper assertions during multi-property verification [33, 27], or to generalize to quantified domains (as elaborated later in §5.1).

When the property is violated, AVR produces a counterexample trace that establishes how to reach a bad state (a state where the property is false) starting from an initial state. AVR prints the counterexample witness in BTOR2 witness format [47], which allows for independent verification of the execution trace using a BTOR2 witness simulator [3]. This allows the designer to debug and pin-point the source of error by analyzing the execution leading to the buggy state.

3.3 Proof Race

AVR supports a variety of configurations and add-on features (as discussed in §3.1). Without detailed knowledge of the input, it is hard to tell upfront which technique will perform the best. Different configurations are useful to tackle different types of problems, though manually trying different configurations can become tedious for the user. To counter this, AVR offers a multi-engine wrapper called *proof race* that automatically runs multiple instances of AVR with different configurations in parallel and offers process-level parallelism. Proof race simply requires giving the input design and resource limits as input, after which it automatically drives multiple AVR working instances until an instance successfully races to the result (while obeying the resource limits).

Such a portfolio-based approach is crucial in practice for fast verification performance since no single technique performs the best in all cases [18, 13], and can be further strengthened by complementing AVR's word-level techniques with state-of-the-art model checking engines like ABC *dprove* [11], IC3ia [20] etc.

4 Case Studies

4.1 Apache Buffer Overflow

We consider patched versions of two buffer-overflow vulnerabilities [37] from standard modules of the Apache web server [1].

apache-escape-absolute corrects a high severity vulnerability `CVE2006-3747` [5] that fixes the out-of-bounds buffer-overflow exploitation which allows a remote attacker to cause a denial of service and execute arbitrary code via crafted URLs. The patched version corrects a check ($c <$ `TOKEN_SZ`) to ($c <$ `TOKEN_SZ` $- 1$).

apache-get-tag fixes a medium severity vulnerability `CVE-2004-0940` [4] that exploits a buffer overflow when copying user-supplied tag strings into finite buffers. A local attacker may leverage this issue to execute arbitrary code on the affected computer with the privileges of the affected Apache server. The patched version corrects a check that validates the length of the tag strings.

In less than a minute, AVR successfully verifies that both of these buffer-overflow exploits are unreachable in the patched versions for *any* buffer size. AVR also provides human-readable proof certificates that are externally verified using Z3, and provides provable assurance against these security vulnerabilities.

4.2 Public Key Authentication Protocol

Needham-Schroeder public key authentication protocol [45] allows establishing mutual authentication between an initiator A and a responder B, after which some session involving the exchange of messages between them can take place. Unfortunately, this protocol is vulnerable to a man-in-the-middle attack [40]. If an intruder I can persuade A to initiate a session with him, he can relay the messages to B and convince B that he is communicating with A.

We consider an instance of the protocol from HWMCC'19 [14, 48] with 3 initiators and responders each, and with an unsafe state defined as a responder being finished authentication with the intruder as a party. Within a minute, AVR finds an execution trace that establishes how to reach an unsafe state. The counterexample witness produced by AVR can be replayed using BtorSIM simulator [3] to verify the execution trace and to debug the protocol.

5 Other Aspects of AVR

5.1 Verifying Distributed Protocols

Beyond verifying model checking problems from the finite domain, AVR has shown preliminary application in the verification of distributed protocols, which are generally expressed over unbounded domains (with an unbounded number of clients, servers, epochs, messages, etc.). The I4 system [42, 41] demonstrates how AVR can be used to verify a simpler finite version of the protocol, followed by generalizing AVR's proof certificates to the unbounded domain. For example, a finite-domain invariant saying "clients c_1 and c_2 cannot both link to the server s" i.e. $\neg(link(c_1, s) \wedge link(c_2, s))$ can be generalized to the unbounded domain as "no two different clients can both link to a server" i.e.
$\forall_{C_1,C_2,S}\ (C_1 \neq C_2) \implies \neg(link(C_1, S) \wedge link(C_2, S))$.

5.2 Strengths

Control-centric properties, where much of the complexity lies in the control logic (like sequential equivalence checking, microprocessor instruction control unit, key-value store) are much easier to verify using AVR. Syntax-guided abstraction hides the domain complexity outside of the problem syntax, and automatically separates important control-flow details from the irrelevant data component. This, combined with data abstraction, allows for scalable model checking with the capacity to scale independent of the bit-width of variables [31, 32].

Push-button verification using AVR eliminates the need for tedious human intervention in verification (like manual identification of abstraction predicates, manually adding helper assertions) by automatic incremental construction of abstraction and word-level clauses using the IC3+SA algorithm.

Provable assurance on the verification outcome is guaranteed by AVR using independently-checkable proof certificates and counterexample traces.

Useful utilities that AVR provides like support for multiple input formats, efficient integration with state-of-the-art SMT solvers, proof race, high-level system statistics, graphical visualizations, etc. altogether aims towards a user-friendly experience without the need for detailed understanding from the part of the user.

5.3 Limitations

Heavy data dependency can make word-level techniques in AVR ineffective for certain problems, especially when a majority of bit-precise values in the data domain play an important role (for example, puzzle solving problems like Tower of Hanoi [36], Peg Solitaire [35], etc. formulated as reachability problems [48]). Logic synthesis and bit-level optimizations [11, 44] can be very useful for such problems and help bit-level checkers to perform better than word-level techniques by significantly decreasing the problem complexity at the bit level.

First-order logic fragments beyond quantifier-free bit-vectors, arrays and uninterpreted functions (like non-linear arithmetic, floating-point numbers, quantifiers, etc.) and properties beyond safety (like *liveness* and *fairness*) have limited support in the current tool implementation. AVR's primarily focus has been on verification of safety properties defined on hardware systems.

6 Conclusions

AVR provides a variety of techniques to efficiently perform automatic word-level verification using SMT solvers with provable guarantees and security. AVR has shown effectiveness in hardware verification [31, 32] (independently evaluated to be the best word-level verifier in the single bit-vector track of HWMCC'19 [14]), and applicability in verifying distributed protocols [41, 42]. In the future, we plan to address some of the current limitations of AVR and extend towards solving practical verification problems beyond the hardware domain.

Acknowledgement. We would like to thank Ranan Fraer, Habeeb Farah and Ziyad Hanna from Cadence Design Systems for their help in shaping some of the concepts presented in this paper.

References

1. Apache HTTP server project. https://httpd.apache.org
2. Btor2Tools. https://github.com/Boolector/btor2tools
3. BtorSIM. https://github.com/Boolector/btor2tools/tree/master/src/btorsim
4. National Vulnerability Database - CVE-2004-0940. https://nvd.nist.gov/vuln/detail/CVE-2004-0940
5. National Vulnerability Database - CVE-2006-3747. https://nvd.nist.gov/vuln/detail/CVE-2006-3747
6. Verification Modulo Theories. http://www.vmt-lib.org
7. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012) pp. 1–1315 (Feb 2018). https://doi.org/10.1109/IEEESTD.2018.8299595
8. Andraus, Z.S., Sakallah, K.A.: Automatic abstraction and verification of verilog models. In: Proceedings of the 41st annual Design Automation Conference. pp. 218–223. ACM (2004)
9. Babić, D., Hu, A.J.: Structural abstraction of software verification conditions. In: International Conference on Computer Aided Verification. pp. 366–378. Springer (2007)
10. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
11. Berkeley Logic Synthesis and Verification Group: ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/ alanmi/abc/ (2017)
12. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bdds. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 193–207. Springer (1999)
13. Biere, A., van Dijk, T., Heljanko, K.: Hardware model checking competition 2017. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 9–9. IEEE (2017)
14. Biere, A., Preiner, M.: Hardware model checking competition (HWMCC) 2019. http://fmv.jku.at/hwmcc19
15. Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to induction-guided abstraction-refinement (ctigar). In: International Conference on Computer Aided Verification. pp. 831–848. Springer (2014)
16. Bradley, A.R.: Sat-based model checking without unrolling. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 70–87. Springer (2011)
17. Burch, J.R., Dill, D.L.: Automatic verification of pipelined microprocessor control. In: International Conference on Computer Aided Verification. pp. 68–80. Springer (1994)
18. Cabodi, G., Loiacono, C., Palena, M., Pasini, P., Patti, D., Quer, S., Vendraminetto, D., Biere, A., Heljanko, K.: Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks. Journal on Satisfiability, Boolean Modeling and Computation **9**, 135–172 (2016)
19. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Ic3 modulo theories via implicit predicate abstraction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 46–61. Springer (2014)

20. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Infinite-state invariant checking with ic3 and predicate abstraction. Formal Methods in System Design **49**(3), 190–218 (2016)
21. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S. (eds.) Proceedings of TACAS. LNCS, vol. 7795. Springer (2013)
22. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: International Conference on Computer Aided Verification. pp. 154–169. Springer (2000)
23. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Progress on the state explosion problem in model checking. In: Informatics. pp. 176–194. Springer (2001)
24. Clarke, E.M., Emerson, E.A., Sifakis, J.: Model checking: algorithmic verification and debugging. Communications of the ACM **52**(11), 74–84 (2009)
25. Clarke Jr, E.M., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Model checking. MIT press (2018)
26. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
27. Dureja, R., Rozier, K.Y.: Fuseic3: An algorithm for checking large design spaces. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 164–171. IEEE (2017)
28. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer-Aided Verification (CAV'2014). Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (July 2014)
29. Een, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design. pp. 125–134. FMCAD Inc (2011)
30. Ghassabani, E., Gacek, A., Whalen, M.W.: Efficient generation of inductive validity cores for safety properties. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 314–325. ACM (2016)
31. Goel, A., Sakallah, K.: Empirical evaluation of ic3-based model checking techniques on verilog rtl designs. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 618–621. IEEE (2019)
32. Goel, A., Sakallah, K.: Model checking of verilog rtl using ic3 with syntax-guided abstraction. In: NASA Formal Methods Symposium. pp. 166–185. Springer (2019)
33. Goldberg, E., Güdemann, M., Kroening, D., Mukherjee, R.: Efficient verification of multi-property designs (the benefit of wrong assumptions). In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 43–48. IEEE (2018)
34. Henzinger, T.A., Qadeer, S., Rajamani, S.K.: You assume, we guarantee: Methodology and case studies. In: International Conference on Computer Aided Verification. pp. 440–451. Springer (1998)
35. Jefferson, C., Miguel, A., Miguel, I., Tarim, S.A.: Modelling and solving english peg solitaire. Computers & Operations Research **33**(10), 2935–2959 (2006)
36. Kotovsky, K., Hayes, J.R., Simon, H.A.: Why are some problems hard? evidence from tower of hanoi. Cognitive psychology **17**(2), 248–294 (1985)
37. Ku, K., Hart, T.E., Chechik, M., Lie, D.: A buffer overflow benchmark for software model checkers. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering. pp. 389–392. ACM (2007)

38. Lee, S., Sakallah, K.A.: Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction. In: International Conference on Computer Aided Verification. pp. 849–865. Springer (2014)
39. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning **40**(1), 1–33 (2008)
40. Lowe, G.: An attack on the needham- schroeder public- key authentication protocol. Information processing letters **56**(3) (1995)
41. Ma, H., Goel, A., Jeannin, J.B., Kapritsos, M., Kasikci, B., Sakallah, K.A.: I4: Incremental inference of inductive invariants for verification of distributed protocols. In: Proceedings of the 27th Symposium on Operating Systems Principles. ACM (2019)
42. Ma, H., Goel, A., Jeannin, J.B., Kapritsos, M., Kasikci, B., Sakallah, K.A.: Towards automatic inference of inductive invariants. In: Proceedings of the Workshop on Hot Topics in Operating Systems. ACM (2019)
43. McMillan, K.L.: Applications of craig interpolants in model checking. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 1–12. Springer (2005)
44. Mishchenko, A., Case, M., Brayton, R., Jang, S.: Scalable and scalably-verifiable sequential synthesis. In: 2008 IEEE/ACM International Conference on Computer-Aided Design. pp. 234–241. IEEE (2008)
45. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Communications of the ACM **21**(12), 993–999 (1978)
46. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 system description. Journal on Satisfiability, Boolean Modeling and Computation **9**, 53–58 (2014 (published 2015))
47. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2, btormc and boolector 3.0. In: International Conference on Computer Aided Verification. pp. 587–595. Springer (2018)
48. Pelánek, R.: Beem: benchmarks for explicit model checkers. In: International SPIN Workshop on Model Checking of Software. pp. 263–267. Springer (2007)
49. Swamy, N., Hriţcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., Bhargavan, K., Fournet, C., Strub, P.Y., Kohlweiss, M., et al.: Dependent types and multi-monadic effects in f. In: ACM SIGPLAN Notices. vol. 51, pp. 256–270. ACM (2016)
50. Wolf, C.: Yosys open synthesis suite. http://www.clifford.at/yosys